Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval*

William McCune

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439-4801

November 6, 1990

Abstract

This article addresses the problem of indexing and retrieving first-order predicate calculus terms in the context of automated deduction programs. The four retrieval operations of concern are to find variants, generalizations, instances, and terms that unify with a given term. Discrimination-tree indexing is reviewed, and several variations are presented. The path-indexing method is also reviewed. Experiments were conducted on large sets of terms to determine how the properties of the terms affect the performance of the two indexing methods. Results of the experiments are presented.

1 Introduction

As automated deduction systems begin to emerge as useful vehicles for studying questions in mathematics and logic, the speed of those systems is becoming more and more important. Given the combinatorial explosion of typical searches, a large speedup in itself will usually not enable programs to find direct proofs of much more difficult theorems. However, reduced response time allows researchers to more easily interact with the program by evaluating failures and trying additional searches with different strategies or axiom sets. Improvement of speed by a factor of ten, five, or even two can easily cause a success.

Careful attention to term indexing in our automated deduction program OTTER [16] has caused substantial speedups. Two different indexing methods are used in OTTER:

*This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

discrimination-tree indexing and path indexing. Initial implementations and experiments showed that some of the indexing operations are usually better performed by discrimination-tree indexing and others by path indexing. In addition, some types of term are much better handled by one or the other of the indexing methods. Thus, no clear winner existed between the two methods. More extensive experiments were then conducted on sets of terms from several application areas. This article contains results of those more extensive experiments.

The problem of term indexing is to maintain a large set of first-order predicate calculus terms and at the same time to provide fast access to members of that set. Four retrieval operations are of interest—to find unifiable terms, generalizations, instances, and alphabetic variants of a query term. In the context of clause-based automated deduction, one wishes to find unifiable terms when applying resolution or paramodulation inference rules or when searching for unit conflict, one wishes to find generalizations when determining whether a new clause is subsumed by an existing clause or when applying rewrite rules, and one wishes to find instances when determining whether a new clause subsumes any existing clauses or when searching for applications of a new rewrite rule. Retrieval of variants can be useful when applying restricted forms of the preceding deduction operations.

Term indexing was used in the early days of automated deduction [17, 10], but it was undocumented or not emphasized in the literature, and the methods are not well known. As a result, it appears that several of the methods were "reinvented". In particular, the origins of the two indexing methods compared in this article remain uncertain. The roots of discrimination-tree indexing appear to be directly in formula manipulation systems, and the roots of path indexing appear to be in database technology.

Discrimination-tree indexing (also called discrimination-net indexing) and some of its variations have appeared in [13, 3, 8, 14, 15, 20, 2, 5, 6]. It is used to find demodulators in [13], it is presented from a Lisp point of view in [3], it is used with very large sets of terms in [15], it is compared to path indexing in [20], it is used in the context of very high-performance Knuth-Bendix completion in [5, 6], and it forms the basis for a high-performance deduction toolkit in [2].

The predecessors of the path-indexing method [20] are coordinate indexing [10] and FPA indexing [17, 12, 11]. Path indexing is a simple but substantial refinement of FPA indexing and coordinate indexing. A hybrid method similar to FPA indexing is presented in [1].

A third class of indexing methods is based on encoding terms into bit strings and using bit operations to aid retrieval [9, 22, 18]. A disadvantage of these methods is that although the bit operations are very fast, in most cases a linear search is required. These methods are not discussed here.

The remainder of the article is divided into preliminaries, including a more precise statement of the indexing problem (Section 2), presentations of discrimination-tree indexing and path indexing (Sections 3 and 4), experiments with the two methods (Section 5), and conclusions.

This article is based on a presentation I gave at the American Association for Artificial Intelligence Spring Series Symposium on High-Performance Theorem Proving at Stanford

University in March 1989.

2 Preliminaries

A term is a variable, a constant, or a complex term. A complex term is a fixed-arity function symbol applied to a sequence of terms. The methods in this article apply as well to atoms (an atom is a relation symbol applied to a sequence of terms). Constants and function symbols are collectively called rigid symbols. Variables are distinguished from constants by starting with a lower-case letter from u to z.

Two terms are unifiable if they have a common instance, in particular, if there exists a substitution of terms for variables that makes the two terms identical. Term t_1 is an instance of term t_2 (and t_2 is a generalization of t_1) if there exists a substitution of terms for variables in t_2 that makes it identical to t_1. Two terms are alphabetic variants if each can be renamed to the other by substituting variables for variables. For all of the unification and matching problems, I assume that the two terms do not share any variables.

When two terms fail to unify or match, I sometimes refer to the reason for the failure. Clashes occur when two rigid symbols cannot be unified or matched. A direct clash can be detected without considering any partial substitution. Indirect clashes and occurs-check failures are detected by considering a partial substitution. Examples of the three types of failure are given in Table 1.

Table 1: Types of Unification Failure

Direct clash	Indirect clash	Occurs-check
$f(a, b)$	$f(x, x)$	$f(x, x)$
$f(a, c)$	$f(a, c)$	$f(y, g(y))$

The problem is to maintain a set of terms and, given a query term that does not share variables with any term in the set, support the following four retrieval operations:

1. Find all terms that unify with the query term, and construct a most general unifying substitution for each.

2. Find all generalizations of the query term, and construct a matching substitution for each.

3. Find all instances of the query term, and construct a matching substitution for each.

4. Find all variants of the query term, and construct a matching substitution for each.